REMARKS/ARGUMENTS

This Amendment is in response to the Office Action dated September 28, 2004. Claims 1-29 are pending. Claims 1-9, 15-21 and 24-29 are rejected. Claims 10-14, 22 and 23 are objected to. Claims 1 and 25 have been amended for clarification. Claim 30 has been added. Accordingly, claims 1-30 remain pending in the present application.

Claim Rejections – 35 USC 103

The Examiner states,

2. Claims 1-3, 9, 15-21, 24, 25, 28, 29 are rejected under 35 U.S.C. 103(a) as being unpatentable over (Witt patent No. 6,240,503) (hereinafter referred to as Witt '503) in view of Brown (patent No. 5,488,730).

3. Witt '503 taught the invention substantially as claimed including a data processing ("DP") system comprising:

(a) Pairs of future state (76) and architectural state pointers (78)(e.g., see fig. 3, col. 17, lines 45-62 and col. 18, lines 39-col. 19, line 6);

(b) Operand queue (84) including at least one entry (e.g., see fig. 3 and col. 18, lines 46-64).

4. Witt '503 did not expressly detail (claims 1, 25) a reference counter associated with each operand queue entry. Brown however taught reference counters associated with each operand entry in queues (e.g., see col. 24, line 59-col. 25, line 10 and col. 26, line 39-col. 27, line 42) Brown also taught operand queue (79)(e.g., see col. 13, lines 54-64).

5. It would have been obvious to one of ordinary skill in the DP art to combine the teachings of Witt '503 and Brown. One of ordinary skill would have been motivated to incorporate the Brown teachings of counters for the operand queues in order to reduce the amount of circuitry needed to keep track of a large number of register dependencies and in particular keep track of multiple dependencies to a single register (e.g., see col. 3, lines 16-52).

Applicant respectfully traverses this rejection. In consideration of the Examiner's rejections, claims 1 and 25 have been amended as follows:

1. An operand file comprising:
at least one pair of future state and architecture state pointers;
an operand queue including at least one entry; and
a reference counter associated with each operand queue entry, whose count indicates the number of registers mapped to the associated operand queue entry.

25. A computer adapted to include an operand file, the operand file comprising:
at least one pair of future state and architecture state pointers;
an operand queue including at least one entry; and
a reference counter associated with each operand queue entry, whose count indicates the number of registers mapped to the associated operand queue entry.

Brown's use of source and destination counts is different than the use in operand file being disclosed. In Brown, the reference counts are used to enforce program order (in-order) execution of instructions, even if some of the instructions can execute correctly out of program order (out-of-order). Specifically, they are used to stall reading source mechanism. When an instruction is decoded, it grabs the snapshot of the source count for its destination register and the snapshots of the destination count for each of its source registers. As the instruction advances through the pipeline, its copy of the source count is decremented as an earlier instruction writes a result to the earlier instruction's destination register that is same as the advancing instruction's source register. Similarly, as the instruction advances through the pipeline, its copy of the destination count is decremented as an earlier instruction reads the earlier instruction's source register that is same as the advancing instruction's destination register. The advancing instruction can read its operand when the operand's destination count is zero and write its result when the result register's source count is zero.

The reference count in operand file is not used for such purpose. The operand file implements register renaming that allows instructions to execute as soon as the source operands and the needed execution units are available, which often results in instructions being executed out-of-order. The operand file uses a mapping scheme involving two pointers for each architectural register, a well-known prior art as the examiner has stated. However, the operand file is not anticipated by the cited reference in that it allows one entry to hold the value of architectural registers, which would happen when an instruction copies the value of one register to another. The reference count in operand file is used to indicate the number of registers mapped to each entry in the operand file, and the count increment and decrement takes place only at the decode and completion stages, respectively. In contrast, Brown's source and destination counts are decremented at various pipeline stages since they are represent the number of register read and write operations.

In Brown, a pair of destination and source counts are associated with each architectural register. In operand file, a reference count is associated with each entry in the operand file that should be viewed as a physical register, as opposed to an architectural register, since register renaming is used. In Brown, a destination count is incremented each time the decoder decodes an instruction that updates the associated architectural register. In operand file, the reference count is incremented once when a free entry is assigned to an architectural register – decoding subsequently another instruction that updates the same architectural register does not increment the same reference count, since another free entry in the operand file is assigned to this second use of the same architectural register. In addition, when a register copy operation is decoded, the reference count associated with the source register (and not the destination register) is incremented in operand file. Also, when an instruction that updates an architectural register completes, the reference count associated with the entry that is mapped to the architectural register before this instruction completes is decremented (and not the entry that is specified by the instruction being completed to be the destination register). Furthermore, since the operand file is a set of physical registers, it can be used to hold values of non-architectural registers, such as immediate values or temporary registers that are used by microcoded sequence of operations.

Therefore, Applicant respectfully disagrees and claims that it is not obvious to combine the teachings of Witt '503 and Brown.

In consideration of the Examiner's rejections, claims 1 and 25 have been amended as follows:

1. An operand file comprising:
at least one pair of future state and architecture state pointers;
an operand queue including at least one entry; and
a reference counter associated with each operand queue entry~~, whose count indicates the number of registers mapped to the associated operand queue entry~~.

25. A computer adapted to include an operand file, the operand file comprising:
at least one pair of future state and architecture state pointers;
an operand queue including at least one entry; and
a reference counter associated with each operand queue entry~~, whose count indicates the number of registers mapped to the associated operand queue entry~~.

6. As per claim 2, 28 Brown taught a free operand entry is assigned to hold a future value of a register by writing the free entry's number into the register's future state pointer and incrementing the free entry's reference count (e.g., see col. 26, line 54-col. 27, line 42).

Applicant respectfully disagrees that Brown teaches the claims 2 and 28, as Brown does not use a register renaming mechanism and that the claims 2 and 28 describe a rename register allocation scheme using operand file. Applicant further requests that claims 2 and 28 be allowed as they are now dependent on amended claims 1 and 25.

7. As per claim 3, 29 Brown taught the assigned entry number is written to the register's architectural state pointer and the reference count of the entry previously assigned to the register is decremented upon completed of the instruction (e.g., see col. 27, line 1-42).

As per claims 9, 18 Brown taught a cancelled instruction does not modify associated architectural state pointer but the reference count of the entry assigned to the register is decremented (e.g., see col. 27, lines 22-42).

Applicant respectfully disagrees that Brown teaches the claims 3 and 29, as Brown does not use a register renaming mechanism and that the claims 3 and 29 describe a rename register deallocation scheme using operand file. Applicant further requests that claims 3 and 29 be allowed as they now are dependent on amended claims 1 and 25.

Applicant also requests that claims 9 and 18 be allowed as they are now dependent on allowed claims.

8. As per claim 15, 16, 17, 19 Witt and Brown did not expressly detail how to process dependencies when at least one operand was immediate. However one of ordinary skill would have been motivated to allocate a entry in the operand queue for an immediate operand (with incrementing/decrementing the counter) at least because a conflict in the destination registers may happen as Brown maintains source and destination queues (e.g., see col. 12, line 58-col. 13, line 15). Also the only time requirement to write the immediate operand would have been to write the operand before the operand was needed to be read (this would have encompassed immediate writing and decrementing and/or delays in writing and/or decrementing that would not conflict with the read).

Applicant respectfully asks that the claims 15, 16, 17, and 19 be allowed as they describe how immediate operands can be handled using operand file, a patented device.

9. As per claim 20, Witt and Brown did not specify how the architectural and future state pointer were handled in a multithreaded implementation. However, since in a multithreaded implementation there would have been the need to separate the registers used in each respective thread from the registers used in another thread for coherency of the data then one of ordinary skill would have been motivated to maintain separate future state and architectural state registers and pointers for each thread and the processor processing the thread.

Operand file efficiently supports creating a child thread that inherits some or all of the register values of the parent thread in a multithreaded processor. As this "inherit" operation requires copying some or all registers from the parent to the child thread in a corresponding manner (that is, register A of the parent thread to register A of the child thread), operand file achieves this operation in one cycle without actually copying the register values. To do this operation using any known prior art would require physically copying the values of the affected registers from the parent thread to the child thread. This would require N number of read ports and N number of write ports in the register file to accomplish it in one cycle. We respectfully disagree that using operand file in multithreaded processors would be obvious to one of ordinary skill in processors. Applicant further requests that claims 20, 21 and 24 be allowed as they are now dependent on allowed claims 3.

10. As per claim 21, Brown taught the storing of value in the future state pointers and architectural state pointers (source and destination queries) without regard to the value of the operand (e.g., see col. 26, line 54-col. 27, line 42).

11. As to the limitations of claim 24, Brown taught that the counters decremented on a

when a entry was remove from the queue and detected a counter value of zero for preventing a read of the values in the queue. This procedure would have required decrementing the counter when the last entry was removed from the queue so that the counter would contain a zero value (e.g., see col. 27, lines 23-43 of Brown).

12. Claims 4-8, 26, 27 are rejected under 35 U.S.C. 103(a) as being unpatentable over Witt '503 in view of Brown as applied to claims 1-3 above, and further in view of Thomas (patent No. 5,535,346).

13. Thomas taught (as to claim 4, 5, 26, 27) each register is assigned a unique operand queue entry upon a reset (e.g., see col. 7, lines 27-48) and all registers that have an undefined value (such as a result of divide by zero) upon reset are assigned to at least one operand queue entry and each of the registers that have defined value upon reset is assigned a unique entry upon a reset (e.g., see col. 7, lines 27-48).

14. It would have been obvious to one of ordinary skill to combine the teachings of Thomas and Witt '503. One of ordinary skill would have been motivated to incorporate the Thomas teachings of a future file control that corrects the future file in a single cycle as least to provide for a more efficient recovery from processing errors or system errors (e.g., see col. 7, lines 27-48 of Thomas).

15. As per claim 6, Thomas taught the entry number previously assigned to the register is obtained from the register's future state pointer (e.g., see col. 2, lines 41-50).

16. As to claim 7, Thomas the entry number previously assigned to the register is obtained from the register's architectural state pointer (e.g., see col. 9, lines 12-67).

17. As to claim 8, Thomas taught in which each the architectural state pointer is copied to its corresponding future pointer when processing an exception condition (e.g., see col. 9, lines 12-67).

Thomas does not teach operand file's use of reference count to indicate the number of registers mapped to a physical register. Applicant therefore requests that claims 4-8, 26, 27 be allowed as they are now dependent on allowed claims.

Accordingly, Applicant respectfully requests reconsideration and allowance of claims 1-29 as now presented.

Allowable Subject Matter

The Examiner states,

18. Claims 10-14, 22, 23 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims.

Applicant appreciates and acknowledges Examiner's indication of allowability of claims

10-14, 22 and 23. Applicant has added new claim 30 which is claim 10 rewritten in independent form. Applicant submits therefore this new claim 30 is allowable over the references.

Conclusion

Applicant respectfully requests allowance and passage to issue of claims 1-30 as now presented. Applicants' attorney believes this application in condition for allowance. Should any unresolved issues remain, Examiner is invited to call Applicants' attorney at the telephone number indicated below.

Respectfully submitted,
SAWYER LAW GROUP LLP

December 28, 2004

[illegible]
Joseph A. Sawyer, Jr.
Attorney for Applicant(s)
Reg. No. 30,801
(650) 493-4540